FFT Analysis of ECG Signals in EDF Format

A Project by

Michael Christian Abay
Juan Wilfredo Ibanez
Mary Julia Segismundo

Submitted to

Luisito L. Agustin
Instructor, ELC 152

In Partial Fulfillment of the Requirements for the Course
ELC 152: Signal Processing

Department of Electronics, Computer and Communications Engineering
School of Science and Engineering
Loyola Schools
Ateneo de Manila University
Quezon City, Philippines

October 2010
Abstract

This project is a particular application to ECG Signals in EDF (European Data Format). The program reads and extracts data from the EDF. After the extraction of data, a Fast Fourier Transform is performed on the data. There are three algorithms for the Fast Fourier Transform (FFT): 1) a recursive FFT 2) a non-recursive FFT 3) an algorithm for computing direct DFT. All three algorithms are compiled in a library. The results of the three FFT algorithms are compared using a separate program that computes the percentage of the error. The output of the program is in CSV file can be opened using an Open Office.
Acknowledgments

The group would like to thank Mr. Ericson Santos and Ms. Catherine Manuela-Lee for their helpful insights on ECG signals which helped us come up with our topic. We would also like to thank Mr. Luisito Agustin for guiding and motivating us to do our project on time. His comments during consultations really helped us a lot to succeed in our project. Lastly, we would also like to express our gratitude to our Almighty Father for the wisdom and all the blessings we received from Him.

For the greater Glory of God.
# Table of Contents

1. Introduction ................................................................................................................... 5  
   1.1. Objectives ....................................................................................................... 5  
   1.2. Significance .................................................................................................... 5  
   1.3. Scope and Limitations .................................................................................... 6  

2. Theoretical Background ................................................................................................. 8  
   2.1. Fourier Transform ........................................................................................... 8  

3. Methodology ................................................................................................................. 10  
   3.1 Test of three implementations of FFT .............................................................. 10  
      3.1.1. DFT Definition ....................................................................................... 10  
      3.1.2. Recursive FFT ...................................................................................... 11  
      3.1.3. Non-recursive FFT .............................................................................. 12  
      3.1.4. Comparison of the three implementations ....................................... 13  
   3.2. Main Program .................................................................................................. 16  
      3.2.1 Extracting EDF ....................................................................................... 16  
      3.2.2 Processing ECG signals in FFT ............................................................ 17  
      3.2.3 CSV Output ............................................................................................ 18  

   4.1 Getting Started ................................................................................................. 19  
   4.2 Using the Program ........................................................................................... 19  

5. Recommendation .......................................................................................................... 20  

Appendices  
   Appendix 1: Program Source Codes ...................................................................... 21  
      A1.1. Main Program ............................................................................................ 21  
      A1.2. fourier.h .................................................................................................... 32  
      A1.3. errorcompute.cpp ................................................................................... 34  
   Appendix 2: Original Project Proposal ................................................................. 35  

Bibliography ...................................................................................................................... 38
1. Introduction

1.1. Objectives

This project aims to create a program that would be able to retrieve raw data samples from an EDF (European Data Format) file and store these samples in memory as an array of values. This array would then be subdivided into sub-arrays or blocks of size 512 samples. Fast Fourier Transform (FFT) is then performed on each block until the whole array is done. An output data file, specifically in Comma-Separated Values (CSV) format, is simultaneously written in the hard-disk as FFT is done to each block.

Additionally, we want to try three different Fourier transform algorithms, namely, the definition of Discrete Fourier Transform (DFT), Recursive FFT, and Non-Recursive FFT, all of which are collected into a single library. We will then compute the percentage difference of both the Recursive and the Non-Recursive FFT against the DFT. The goal here is to find out which of the two algorithms would have the least error.

1.2. Significance

Our program is significant because it will allow us to explore the EDF format and how it can be accessed. This normally requires the knowledge on the file format itself and how information is organized within the file. File specifications are easily found in its official website in the internet.

Additionally, it allows us to explore different Fourier transform algorithms and compare the accuracy between one algorithm against the DFT definition. Our program is useful for students who are just starting to understand Fourier transform and who want to know how it can be concretely applied to certain contexts, here specifically for ECG records.
Normally, ECG records are inspected by doctors by looking at the ECG waveform in the time domain, and measuring certain parameters, for example, R-R interval, QRS complex, etc. A diagnosis is arrived at when the doctor has found abnormalities in these measurements.

In the past years, there has been great interest in exploring the possibility of studying the ECG waveform in the frequency domain. Some studies have found that abnormalities in the sinus rhythm would tend to decrease the frequency value with the highest magnitude. For example, for normal sinus rhythm, the frequency of highest magnitude is around the vicinity of 7Hz. With some abnormalities, this value would tend to decrease. These kinds of studies pose a promising future for the frequency-domain analysis of ECG records, and it could potentially revolutionize the way heart abnormalities are diagnosed. These studies are only possible by performing Fourier transform analyses on ECG signals.

1.3. Scope and Limitations

Two different EDF versions are in use today according to the format’s official site in the internet. There is the basic EDF file format and the EDF+ file format. The only notable advantage of the EDF+ format is that it integrates an annotation channel in the file aside from the basic contents of the original EDF format. The basic contents include the header, signal and main data.

In our program we only used the basic EDF version as we are not interested in the contents of the annotations part of the file. We only need some basic information from the header of the file, i.e. sampling rate, no. of signals, length of data, etc. to be used to access the raw data values and to perform the FFT.

It is possible for an EDF file to have more than one signal, i.e. V1, V2, V3, etc. A limitation in our program is that we can only access an EDF file that only has a single signal. Our program is not able to process those with two or more signals.
The EDF files we have used to test our program were retrieved from the Physionet Databank (www.physionet.org). More specifically, we have used the databank for CU Ventricular Tachyarrhythmia because it contains basic EDF files each with single signals.

The library of Fourier transform functions we made contains three different algorithms: the definition of Discrete Fourier Transform (DFT), Recursive FFT, and Non-Recursive FFT.

We have studied the accuracy in terms of percentage error between both FFT algorithms (recursive and non-recursive) against DFT. However, we have not tested the speed or number of computations per algorithm.
2. Theoretical Background

2.1 Fourier Transform

Fourier Transform can be described as a mathematical operation which usually involves conversion of a time domain to frequency domain function. There are two types of Fourier Transform – Discrete Fourier Transform and Fast Fourier Transform. In Discrete Fourier Transform or DFT, equation(1) will be helpful in understanding the concepts on it. In equation (1), x(n) refers to the input signal amplitude in time domain while X(k) is the output obtained in frequency domain. It should also be noted that N is the number of samples.

\[
X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N} \tag{1}
\]

Equation (1) can be expressed in another form with the idea that \( W_N = e^{-j2\pi/n} \). Hence, equation (2) will be obtained.

\[
X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} \tag{2}
\]

On the other hand, Fast Fourier Transform or FFT is a fourier transform that can handle the mathematical operation of transforming functions in a much faster way than DFT. A proof to this is that in DFT it takes around \( 2N^2 \) number of operation steps in DFT where N is the number of samples. While in FFT, it only takes around \( 2N\log_2N \). Clearly, it can be observed from there that the number of operational steps is reduced in FFT. From an intensive mathematical procedure, FFT is able to make the procedure more efficient. There are also different algorithms under FFT. In our project, we are dealing with Radix-2 FFT for the FFT analysis of the ECG signals.
Radix-2 FFT is one of the algorithms of FFT wherein the N samples is a power of 2 which adheres to this: \( N = 2^s \). The reason why it is also called such is that there is two parts in performing the algorithm: odd and even sample. This can be explained by the equations below.

\[
X(k) = \sum_{m=0}^{(N/2)-1} x(2m)W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m + 1)W_N^{(2m+1)k} \tag{3}
\]

Equation (3) was derived from equation (2) wherein there is a dichotomy in the equation referring to the even and odd part. Furthermore, this was done with the note that \( f_1(m) = x(2m) \) and \( f_2(m) = x(2m + 1) \). From that, the even and odd numbered samples are more visualized. We can still simply this by equating the first term with \( F_1(k) \). It should also be realized that the second term can also be expressed in the same way but with an additional factor. Through computation, we had this equivalent form for the second term: \( W_N^k F_2(k) \). There is an additional factor of \( W_N^k \) which is called the twiddle factor. Combining the two will result to equation (4).

\[
X(k) = F_1(k) + W_N^k F_2(k) \tag{4}
\]
3. Methodology

3.1 Test of three implementations of FFT

In our group, we decided to create a library for the three different implementation of FFT. The purpose of this is to compare the recursive and non-recursive FFT with the direct DFT. The library code for this has this filename: fourier.f. The library consists of three parts: DFT definition, recursive FFT and non-recursive FFT.

3.1.1. DFT Definition

```cpp
complex<float> *dft(int N,complex<float> *f)
{
    complex<float> *F = new complex<float>[N];
    complex<float> e;
    for(int k=0;k<N;k++)
    {
        F[k] = (0,0);
        for(int n=0;n<N;n++)
        {
            e = complex<float>(cos(2*PI*k*n/N),sin(-2*PI*k*n/N)); //compute exponential factor
            F[k] = F[k] + f[n]*e;
        }
    }
    return F;
}
```

In the first part of the library, we considered the definition of DFT which is shown in equation (1). This equation reflects in the code through

\[ e = \text{complex<float}>(\cos(2\pi k^n/N),\sin(-2\pi k^n/N)); \]

//compute exponential factor
\[ F[k] = F[k] + f[n]*e; \]

Basically, after getting the necessary data from the edf, the program will have to perform the DFT operation on the said input.
3.1.2. Recursive FFT

```c
complex<float> *fft_rec(int N, complex<float> *f)
{
    int Nprime, n, kprime;
    complex<float> *fe = new complex<float>[N/2];
    complex<float> *fo = new complex<float>[N/2];
    complex<float> *Fe = new complex<float>[N/2];
    complex<float> *Fo = new complex<float>[N/2];
    complex<float> *F = new complex<float>[N];;
    complex<float> twiddle;
    if(N==1)
    {
        return f; //trivial if N==1
    }
    else
    {
        //perform 2 sub-transforms
        Nprime = N/2; //size of sub-transforms
        for(int n=0; n<Nprime; n++)
        {
            twiddle = complex<float>(cos(2*PI*n/N),sin(-2*PI*n/N));
            fe[n] = f[n]+f[n+Nprime]; //even subset
            fo[n] = (f[n]-f[n+Nprime])*twiddle; //odd subset
        }
        Fe = fft_rec(Nprime,fe); //even k
        Fo = fft_rec(Nprime,fo); //odd k
        for(int kprime=0; kprime<Nprime; kprime++)
        {
            F[2*kprime] = Fe[kprime]; //even k
            F[2*kprime+1] = Fo[kprime]; //odd k
        }
        return F;
    }
}
```

The second part of the library deals with recursive FFT. In this case, it already uses the Radix-2 FFT algorithm. Hence, the operation is faster and more efficient. It can also be noticed in that part that the odd and even indexes were separated. Also there is a twiddle factor being multiplied to the odd part. An interesting note here is that there is this butterfly effect as can be seen in the comment part of the code. It is mentioned in our research that this reduces the number of operations. The effect of this is that it makes the recursion faster. In recursion also, the function is called within a function.
3.1.3. Non-Recursive FFT

complex<float> *fft_nrec(int N, complex<float> *f) {
    int d;
    int M = N/2;
    float w, z;
    complex<float> DFTe(0,0);
    complex<float> DFTo(0,0);
    complex<float> *F = new complex<float>[N];
    complex<float> e(0,0);
    complex<float> i(0,1);
    complex<float> g(0,0);
    for(int k = 0; k < N; k++) {
        w = (-2.00*PI)/M;
        DFTe = complex<float>(0,0);
        DFTo = complex<float>(0,0);
        for(int m = 0; m < M; m++) {
            e = exp(w*m*k*i);
            d = 2*m;
            if(m == 0) { DFTe = f[d]*e; }
            else {
                DFTe = DFTe + f[d]*e;
            }
        }
        for(int m = 0; (2*M == N) && m < M; m++) {
            e = exp(w*m*k*i);
            d = 2*m+1;
            if(m == 0) { DFTo = f[d]*e; }
            else {
                DFTo = DFTo + f[d]*e;
            }
        }
        for(int m = 0; (2*M != N) && m < (M-1); m++) {
            e = exp(w*m*k*i);
            d = 2*m+1;
            if(m == 0) { DFTo = f[d]*e; }
            else {
                DFTo = DFTo + f[d]*e;
            }
        }
        z = (-2.00*PI)/N;
        g = exp(z*k*i);
        F[k] = DFTe + (DFTo*g);
    }
    return F;
}

For the non-recursive FFT, the algorithm happens in place. Moreover, the other characteristic is that it usually uses for loops. In the last portion of the library, it can be observed that there is a nested for loop in a for loop. After getting the fourier transform of the odd and even part, it will add both in order to get the fourier transform of the input.
3.1.4 Comparison of the three implementations

Figure 3.1: DFT

Figure 3.2: Recursive FFT
What we did next was to compare the FFT of the three implementations. Graphically, it can be noticed that the three of them look the same. We also determine the percent difference in terms of the points in the graph. In general, it shows a very minimum percent difference. The data from Table 3.1 and Table 3.2 were obtained from the same chunk of 512 samples. The complete data can be found in the soft copy.

<table>
<thead>
<tr>
<th></th>
<th>DFT</th>
<th>Recursive</th>
<th>% Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.12</td>
<td>26.12</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>21.4748</td>
<td>21.4748</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>25.5715</td>
<td>25.5715</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>78.4249</td>
<td>78.4249</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9.56284</td>
<td>9.56284</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>20.2809</td>
<td>20.2809</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>56.1174</td>
<td>56.1174</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>21.5288</td>
<td>21.5288</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>45.8588</td>
<td>45.8588</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>55.9778</td>
<td>55.9778</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>33.0658</td>
<td>33.0658</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>80.1706</td>
<td>80.1706</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>36.3517</td>
<td>36.3517</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>15.0778</td>
<td>15.0778</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: 0 % Difference
Table 3.2: With non-zero % difference

<table>
<thead>
<tr>
<th>DFT</th>
<th>Recursive</th>
<th>% Diff</th>
<th>DFT</th>
<th>NRecursive</th>
<th>% Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.08216</td>
<td>3.08217</td>
<td>0</td>
<td>3.08216</td>
<td>3.08209</td>
<td>0</td>
</tr>
<tr>
<td>4.51124</td>
<td>4.51124</td>
<td>0</td>
<td>4.51124</td>
<td>4.51114</td>
<td>0</td>
</tr>
<tr>
<td>1.71225</td>
<td>1.71235</td>
<td>0.01</td>
<td>1.71225</td>
<td>1.71227</td>
<td>0</td>
</tr>
<tr>
<td>1.64362</td>
<td>1.64367</td>
<td>0</td>
<td>1.64362</td>
<td>1.64366</td>
<td>0</td>
</tr>
<tr>
<td>4.23716</td>
<td>4.23712</td>
<td>0</td>
<td>4.23716</td>
<td>4.23721</td>
<td>0</td>
</tr>
<tr>
<td>0.796126</td>
<td>0.796093</td>
<td>0</td>
<td>0.796128</td>
<td>0.796081</td>
<td>-0.01</td>
</tr>
<tr>
<td>0.091682</td>
<td>0.091659</td>
<td>-0.03</td>
<td>0.091682</td>
<td>0.0917382</td>
<td>0.06</td>
</tr>
<tr>
<td>3.67942</td>
<td>3.67938</td>
<td>0</td>
<td>3.67942</td>
<td>3.67934</td>
<td>0</td>
</tr>
<tr>
<td>0.901719</td>
<td>0.901732</td>
<td>0</td>
<td>0.901719</td>
<td>0.901582</td>
<td>0</td>
</tr>
<tr>
<td>0.780095</td>
<td>0.780148</td>
<td>0.01</td>
<td>0.780095</td>
<td>0.780121</td>
<td>0</td>
</tr>
<tr>
<td>1.90867</td>
<td>1.90856</td>
<td>-0.01</td>
<td>1.90867</td>
<td>1.90866</td>
<td>0</td>
</tr>
<tr>
<td>0.53445</td>
<td>0.534521</td>
<td>0.01</td>
<td>0.53445</td>
<td>0.534463</td>
<td>0.01</td>
</tr>
<tr>
<td>0.588357</td>
<td>0.588368</td>
<td>0</td>
<td>0.588357</td>
<td>0.58828</td>
<td>-0.01</td>
</tr>
<tr>
<td>0.309664</td>
<td>0.309663</td>
<td>0.01</td>
<td>0.309664</td>
<td>0.309606</td>
<td>-0.02</td>
</tr>
<tr>
<td>0.374687</td>
<td>0.374643</td>
<td>-0.01</td>
<td>0.374687</td>
<td>0.37476</td>
<td>0</td>
</tr>
<tr>
<td>0.557028</td>
<td>0.556977</td>
<td>-0.01</td>
<td>0.557028</td>
<td>0.557047</td>
<td>0</td>
</tr>
<tr>
<td>0.57298</td>
<td>0.572803</td>
<td>-0.03</td>
<td>0.57298</td>
<td>0.572993</td>
<td>0</td>
</tr>
<tr>
<td>0.052551</td>
<td>0.052571</td>
<td>0.04</td>
<td>0.052551</td>
<td>0.0525145</td>
<td>-0.07</td>
</tr>
<tr>
<td>0.649134</td>
<td>0.649225</td>
<td>0.01</td>
<td>0.649134</td>
<td>0.649101</td>
<td>-0.01</td>
</tr>
<tr>
<td>0.711859</td>
<td>0.711872</td>
<td>0</td>
<td>0.711859</td>
<td>0.711869</td>
<td>0</td>
</tr>
<tr>
<td>0.664123</td>
<td>0.664122</td>
<td>0</td>
<td>0.664123</td>
<td>0.664097</td>
<td>0</td>
</tr>
<tr>
<td>1.7195</td>
<td>1.71948</td>
<td>0</td>
<td>1.7195</td>
<td>1.71953</td>
<td>0</td>
</tr>
<tr>
<td>0.362998</td>
<td>0.362963</td>
<td>-0.04</td>
<td>0.362998</td>
<td>0.362905</td>
<td>-0.03</td>
</tr>
</tbody>
</table>

At the same time we also got the average percent difference:

Table 3.3: Average % Difference

<table>
<thead>
<tr>
<th>With respect to DFT Definition</th>
<th>% Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ave Error for Recursive</td>
<td>0.027129</td>
</tr>
<tr>
<td>Ave Error for Non-Recursive</td>
<td>0.018379</td>
</tr>
</tbody>
</table>

The table gives a summary that in fact the percent difference after analyzing the different implementation is very minimal.
3.2 Main Program

There are three main parts of the main program. First is the extraction of data values from the EDF file. Second is the application of the Fourier transform. Last is the printing of the frequency-domain values in CSV file format.

3.2.1 Extracting EDF

In order to effectively and efficiently retrieve data samples from an EDF file, we have used an open-source program written by Teunis van Beelen, one of the pioneers of the EDF version. This program and all relevant documentation can be downloaded from <http://www.teuniz.net/edf2ascii/>.

We edited most of the program and fashion it to our purpose. Originally, the code is composed of 829 lines. We changed parts of the code so that instead of writing text files in the hard disk, it only stores the converted data values in memory to be processed later using our Fourier functions.

To do this, we first found the part of the code where it converts the contents of the data into numeric data.

```c
if((d_tmp<(time_tmp+0.00000000000001))&&(d_tmp>(time_tmp-0.00000000000001))&&(edfparam[j].smp_written<edfparam[j].smp_per_record))
{
fprintf(outputfile, ",%f", value_tmp);
edfparam[j].smp_written++;
}
```

We then added a line in by which the numeric value (value_tmp) is stored in a dynamically allocated array (dynamicArray) whose size is equal to the sample size of the whole edf file.
if((d_tmp<(time_tmp+0.00000000000001))&&(d_tmp>(time_tmp-
0.00000000000001))&&(edfparam[j].smp_written<edfparam[j].smp_per_record))
{
    value_tmp = ((*(signed short *)cnv_buf)+edfparam[j].buf_offset+edfparam[j].smp_written) + edfparam[j].offset) *
edfparam[j].sense;
    dynamicArray[(i*recordsize)+edfparam[j].smp_written] = value_tmp; //put value_tmp to dynamicArray
    edfparam[j].smp_written++;
}

3.2.2 Processing ECG signals in FFT

Since dynamicArray is a one-dimesional array and we want to divide it into blocks
of size 512, there is a need to create a second array and copy the contents onto it the
contents of the dynamicArray. This array (array512) is a dynamically allocated two-
dimesional array. It is composed of 512-sized sub-arrays. The number of sub-arrays is
equal to (dynamicArray/ 512).

int numArrays = 0;
umArrays = (datarecords*recordsize)/512;

//Initialize 2-dimensional array512
complex<float> **array512 = new complex<float>*[numArrays];

//Initialize sub-arrays of array512 with 512 block-size
for(int i = 0; i < numArrays; i++)
{
    array512[i] = new complex<float>[512];
}

//Copy contents of dynamicArray to array512
int ctr = 0;
for(int j=0;j<numArrays && ctr<numSamples;j++)
{
    for(int k=0;k<512 && ctr<numSamples;k++)
    {
        array512[j][k] = dynamicArray[ctr];
        ctr++;
    }
}
3.2.3 CSV output

After copying all the contents from the dynamicArray to array512, FFT is then applied to array512 by blocks or sub-arrays. For the main program, we have use the function fft_rec. This means that the fft implemented is the one with recursion. The Fourier transform output for each block is then printed on a CSV file on the same directory as the main program.

//Perform FFT for each 512-size block
ofstream outStream;
outStream.open("fft.csv");
complex<float> *newArray = new complex<float>[512];

float incfreq = (recordsize/data_record_duration)/512.000;
float frequency = 0.000;
for(int i=0;i<numArrays;i++)
{
    newArray = fft_rec(512, array512[i]);
    outStream << "Array " << i+1 << "n";
    for(int j=0;j<256;j++)
    {
        frequency = incfreq*j;
        outStream << frequency << ",
        outStream << abs(newArray[j]) << "n";
    }
}
outStream.close();

After the Fourier transform of the whole array is finished, all dynamically allocated arrays are deleted.

//delete dynamic arrays;
delete [] newArray;
delete [] dynamicArray;
for(int i=0;i<numArrays;i++)
{
    delete [] array512[i];
}
delete [] array512;

4.1 Getting Started

In this program, it is important to note that the input should have a filename extension of .edf in order for it to be implemented. As mentioned in the earlier part, this deals with ecg signals. Furthermore, the program gets FFT of the input using Radix-2 algorithm in our program code.

4.2 Using the Program

The program is run by dragging an EDF file into the executable file, or through the use of MSDOS by typing: edf2ascii <edf file>. The program produces four output text files: header, annotation, signal and data. For our purpose, we are only concerned on the data part and some information found in the header and signal parts. While the annotations part is unused since we are using the basic EDF version. The one with the .csv filename extension will be used. This file will contain two columns of numbers. The first one is the values for the magnitude while the other one will be values for the frequency domain. In order to see that, the user will have to open it in an Open Office Application and perform there the part of the data.
5. Recommendation

Our group was able to succeed in achieving our aim. However, we also know that there are also room for improvements in some of the parts of our project. One of this is to explore on the possibility of creating a Graphic User Interface for aesthetics and efficiency. In the future, other groups can also implement a code that can also process additional file aside from those in EDF. In that way, the program can be more flexible.
Appendix 1: Program Source Codes

A1.1. Main Program

#include <iostream>
#include <fstream>
#include <complex>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include "fourier.h"
using namespace std;

struct edfparamblock{
    int smp_per_record;
    int smp_written;
    int dig_min;
    int dig_max;
    int offset;
    int buf_offset;
    double phys_min;
    double phys_max;
    double time_step;
    double sense;
} *edfparam;

int main(int argc, char *argv[])
{
    FILE *inputfile,
    *outputfile,
    *annotationfile;

    const char *fileName;

    int i, j, k, p, r, n,
    pathlen,
    fname_len,
    signals,
    datarecords,
    datarecordswritten,
    recordsize,
    recordfull,
    edfplus,
    annot_ch[256],
    nr_annot_chns,
    skip,
    max,
    onset,
    duration,
    zero,
    max_tal_ln;

    char path[512],
    ascii_path[512],
    *edf_hdr,
    *scratchpad,
    *cnv_buf,
    *time_in_txt,
    *duration_in_txt;

    double data_record_duration,
    elapsedtime,
if(argc!=2)
{
 printf("\nEDF(+) to ASCII converter version 1.2\n"
  "Copyright 2007 Teunis van Beelen\n"
  "teuniz@gmail.com\n"
  "Usage: edf2ascii <filename>\n\n");
 return(1);
}

strcpy(path, argv[1]);
strcpy(ascii_path, argv[1]);

pathlen = strlen(path);
if(pathlen<5)
{
 printf("Error, filename must contain at least five characters.\n");
 return(1);
}

scratchpad = (char *)malloc(128);
if(scratchpad==NULL)
{
 printf("Malloc error! (scratchpad).\n");
 return(1);
}

fname_len = 0;
for(i=pathlen; i>0; i--)
{
 if((path[i-1]=='/')||(path[i-1]=='\'))  break;
  fname_len++;
}
fileName = path + pathlen - fname_len;

for(i=0; fileName[i]!=0; i++);
if(i==0)
{
 printf("Error, filename must contain at least five characters.\n");
 return(1);
}

i -= 4;
if((strcmp((const char *)fileName + i, ".edf"))&&(strcmp((const char *)fileName + i, ".EDF")))
{
 printf("Error, filename extension must have the form ".edf" or ".EDF\n");
 return(1);
}

/***************** check header **********************/
in
inputfile = fopen(path, "rb");
if(inputfile==NULL)
{
 printf("Error, can not open file %s for reading\n", path);
 return(1);
}

if(fseek(inputfile, 0xfc, SEEK_SET))
{
 printf("Error, reading file %s\n", path);
 fclose(inputfile);
 return(1);
}
if(fread(scratchpad, 4, 1, inputfile)!=1)
{
    printf("Error, reading file %s\n", path);
    fclose(inputfile);
    return(1);
}

scratchpad[4] = 0;
signals = atoi(scratchpad);
if((signals<1)||(signals>256))
{
    printf("Error, number of signals in header is %i\n", signals);
    fclose(inputfile);
    return(1);
}

edf_hdr = (char *)malloc((signals + 1) * 256);
if(edf_hdr==NULL)
{
    printf("Malloc error! (edf_hdr)\n");
    fclose(inputfile);
    return(1);
}

rewind(inputfile);
if(fread(edf_hdr, (signals + 1) * 256, 1, inputfile)!=1)
{
    printf("Error, reading file %s\n", path);
    fclose(inputfile);
    free(edf_hdr);
    return(1);
}

for(i=0; i<((signals+1)*256); i++)
{
    if(edf_hdr[i]==',') edf_hdr[i] = '\''; /* replace all comma's in header by single quotes because they */
    /* interfere with the comma-separated txt-files */
}
if(strncmp(edf_hdr, "0       ", 8))
{
    printf("Error, EDF-header has unknown version\n");
    fclose(inputfile);
    free(edf_hdr);
    return(1);
}

strncpy(scratchpad, edf_hdr + 0xec, 8);
scratchpad[8] = 0;
datarecords = atoi(scratchpad);
if(datarecords<1)
{
    printf("Error, number of datarecords in header is %i\n", datarecords);
    fclose(inputfile);
    free(edf_hdr);
    return(1);
}

strncpy(scratchpad, edf_hdr + 0xf4, 8);
scratchpad[8] = 0;
data_record_duration = atof(scratchpad);

nr_annot_chns = 0;

if((strncmp(edf_hdr + 0xc0, "EDF+C ", 10))&&
   (strncmp(edf_hdr + 0xc0, "EDF+D ", 10)))
{
    edfplus = 0;
}
else
{
edfplus = 1;
for(i=0; i<signals; i++)
{
    if(!(strncmp(edf_hdr + 256 + i * 16, "EDF Annotations ", 16)))
    {
        annot_ch[nr_annot_chns] = i;
        nr_annot_chns++;
        if(nr_annot_chns>255) break;
    }
}
if(!nr_annot_chns)
{
    printf("Error, file is marked as EDF+ but it has no annotations.\n");
    fclose(inputfile);
    free(edf_hdr);
    return(1);
}
edfparam = (struct edfparamblock *)malloc(signals * sizeof(struct edfparamblock));
if(edfparam==NULL)
{
    printf("Malloc error! (edfparam)\n");
    fclose(inputfile);
    free(edf_hdr);
    return(1);
}
recordsize = 0;
for(i=0; i<1; i++)
{
    strncpy(scratchpad, edf_hdr + 256 + signals * 216 + i * 8, 8);
    scratchpad[8] = 0;
    edfparam[i].smp_per_record = atoi(scratchpad);
    edfparam[i].buf_offset = recordsize;
    recordsize += edfparam[i].smp_per_record;
    strncpy(scratchpad, edf_hdr + 256 + signals * 104 + i * 8, 8);
    scratchpad[8] = 0;
    edfparam[i].phys_min = atof(scratchpad);
    strncpy(scratchpad, edf_hdr + 256 + signals * 112 + i * 8, 8);
    scratchpad[8] = 0;
    edfparam[i].phys_max = atof(scratchpad);
    strncpy(scratchpad, edf_hdr + 256 + signals * 120 + i * 8, 8);
    scratchpad[8] = 0;
    edfparam[i].dig_min = atoi(scratchpad);
    strncpy(scratchpad, edf_hdr + 256 + signals * 128 + i * 8, 8);
    scratchpad[8] = 0;
    edfparam[i].dig_max = atoi(scratchpad);
    edfparam[i].time_step = data_record_duration / edfparam[i].smp_per_record;
    edfparam[i].sense = (edfparam[i].phys_max - edfparam[i].phys_min) / (edfparam[i].dig_max - edfparam[i].dig_min);
    edfparam[i].offset = edfparam[i].phys_max / edfparam[i].sense - edfparam[i].dig_max;
}
}
cnv_buf = (char *)malloc(recordsize * 2);
if(cnv_buf==NULL)
{
    printf("Malloc error! (cnv_buf)\n");
    fclose(inputfile);
    free(edf_hdr);
    free(edfparam);
    return(1);
}
free(scratchpad);
max_tal_ln = 0;

for(r=0; r<nr_annot_chns; r++)
{
    if(max_tal_ln<edfparam[annot_ch[r]].smp_per_record * 2)  max_tal_ln = edfparam[annot_ch[r]].smp_per_record * 2;
}

if(max_tal_ln<128)  max_tal_ln = 128;

scratchpad = (char *)malloc(max_tal_ln + 2);
if(scratchpad==NULL)
{
    printf("Malloc error! (scratchpad)\n");
    fclose(inputfile);
    free(cnv_buf);
    free(edf_hdr);
    free(edfparam);
    return(1);
}

duration_in_txt = (char *)malloc(max_tal_ln + 2);
if(duration_in_txt==NULL)
{
    printf("Malloc error! (duration_in_txt)\n");
    fclose(inputfile);
    free(scratchpad);
    free(cnv_buf);
    free(edf_hdr);
    free(edfparam);
    return(1);
}

time_in_txt = (char *)malloc(max_tal_ln + 2);
if(time_in_txt==NULL)
{
    printf("Malloc error! (time_in_txt)\n");
    fclose(inputfile);
    free(duration_in_txt);
    free(scratchpad);
    free(cnv_buf);
    free(edf_hdr);
    free(edfparam);
    return(1);
}

/************************* write data ******************************/

ascii_path[pathlen-4] = 0;
strcat(ascii_path, "_data.csv");
outputfile = fopen(ascii_path, "wb");
if(outputfile==NULL)
{
    printf("Error, can not open file %s for writing\n", ascii_path);
    fclose(inputfile);
    fclose(annotationfile);
    free(edf_hdr);
    free(edfparam);
    free(cnv_buf);
    free(time_in_txt);
    free(duration_in_txt);
    free(scratchpad);
    return(1);
}

fprintf(outputfile, "Time");

for(i=0; i<(signals-nr_annot_chns); i++)
{  fprintf(outputfile, ",%i", i + 1);
}

if(fputc(\n', outputfile)==EOF)
{
  printf("Error when writing to outputfile\n");
  fclose(inputfile);
  fclose(annotationfile);
  fclose(outputfile);
  free(edf_hdr);
  free(edfparam);
  free(cnv_buf);
  free(time_in_txt);
  free(duration_in_txt);
  free(scratchpad);
  return(1);
}

if(fseek(inputfile, (signals + 1) * 256, SEEK_SET))
{
  printf("Error when reading inputfile\n");
  fclose(inputfile);
  fclose(annotationfile);
  fclose(outputfile);
  free(edf_hdr);
  free(edfparam);
  free(cnv_buf);
  free(time_in_txt);
  free(duration_in_txt);
  free(scratchpad);
  return(1);
}

/***************** start data conversion ***********************/

datarecordswritten = 0;

cout << "Data Records: " << datarecords << "\n";
cout << "Record Size: " << recordsize << "\n";

int numSamples = datarecords*recordsize;

dynamicArray = new float[numSamples];

for(i=0; i<datarecords; i++)
{
  for(j=0; j<1; j++)  edfparam[j].smp_written = 0; //j<1 replaces j<signals

  if(fread(cnv_buf, recordsize * 2, 1, inputfile)!=1)
  {
    printf("Error when reading inputfile during conversion\n");
    fclose(inputfile);
    fclose(annotationfile);
    fclose(outputfile);
    free(edf_hdr);
    free(edfparam);
    free(cnv_buf);
    free(time_in_txt);
    free(duration_in_txt);
    free(scratchpad);
    return(1);
  }

  if(edfplus)
  {
    max = edfparam[annot_ch[0]].smp_per_record * 2;
p = edfparam[annot_ch[0]].buf_offset * 2;
/* extract time from datarecord */
for(k=0; k<max; k++)
{
  if(k>max_tal_ln)
  {
    printf("Error, TAL in record %i exceeds my buffer\n", datarecordswritten + 1);
    fclose(inputfile);
    fclose(annotationfile);
    fclose(outputfile);
    free(edf_hdr);
    free(edfparam);
    free(cnv_buf);
    free(time_in_txt);
    free(duration_in_txt);
    free(scratchpad);
    return(1);
  }
  scratchpad[k] = cnv_buf[p + k];
  if(scratchpad[k]==20) break;
}
scratchpad[k] = 0;
elapsedtime = atof(scratchpad);
/* process annotations */
for(r=0; r<nr_annot_chns; r++)
{
  p = edfparam[annot_ch[r]].buf_offset * 2;
  max = edfparam[annot_ch[r]].smp_per_record * 2;
  n = 0;
  zero = 0;
  onset = 0;
  duration = 0;
  time_in_txt[0] = 0;
  duration_in_txt[0] = 0;
  scratchpad[0] = 0;
  for(k=0; k<max; k++)
  {
    if(k>max_tal_ln)
    {
      printf("Error, TAL in record %i exceeds my buffer\n", datarecordswritten + 1);
      fclose(inputfile);
      fclose(annotationfile);
      fclose(outputfile);
      free(edf_hdr);
      free(edfparam);
      free(cnv_buf);
      free(time_in_txt);
      free(duration_in_txt);
      free(scratchpad);
      return(1);
    }
    scratchpad[n] = cnv_buf[p + k];
    if((scratchpad[n]!=20)&&(scratchpad[n]!=21)&&(scratchpad[n]!=0))
    {
      if((scratchpad[n]>126)||(scratchpad[n]<32))  scratchpad[n] = '.';
      /* replace non-readable ASCII-character by dot */
      if(scratchpad[n]==',')  scratchpad[n] = '\';
      /* replace comma by single quote ' */
    }
    if(scratchpad[n]==0)
    {
      n = 0;
    }
onset = 0;
duration = 0;
time_in_txt[0] = 0;
duration_in_txt[0] = 0;
scratchpad[0] = 0;
zero++; continue;
} else zero = 0;

if(zero>1) break;

if(scratchpad[n]==20)
{
 if(duration)
 {
 scratchpad[n] = 0;
 strcpy(duration_in_txt, scratchpad);
 n = 0;
 duration = 0;
 scratchpad[0] = 0;
 continue;
 }
 else if(onset)
 {
 scratchpad[n] = 0;
 if(n) fprintf(annotationfile, "%s,%s,%s\n", time_in_txt, duration_in_txt, scratchpad);
 n = 0;
 duration = 0;
 duration_in_txt[0] = 0;
 scratchpad[0] = 0;
 continue;
 }
 else
 {
 scratchpad[n] = 0;
 strcpy(time_in_txt, scratchpad);
 n = 0;
 onset = 1;
 duration = 0;
 duration_in_txt[0] = 0;
 scratchpad[0] = 0;
 continue;
 }
}

if(scratchpad[n]==21)
{
 if(onset)
 {
 scratchpad[n] = 0;
 strcpy(time_in_txt, scratchpad);
 onset = 1;
 } else
 {
 scratchpad[n] = 0;
 strcpy(time_in_txt, scratchpad);
 n = 0;
 duration = 1;
 duration_in_txt[0] = 0;
 scratchpad[0] = 0;
 continue;
 }

if(++n>max_tal_ln)
{
 printf("Error, TAL in record %i exceeds my buffer\n", datarecordwritten + 1);
 fclose(inputfile);
 fclose(annotationfile);
 fclose(outputfile);
 free(edf_hdr);
}
free(edfparam);
free(cnv_buf);
free(time_in_txt);
free(duration_in_txt);
free(scratchpad);
return(1);
}
} else elapsedtime = datarecordswritten * data_record_duration;

/* done with timekeeping and annotations, continue with the data */
do {
    time_tmp = 10000000000.0;
    for(j=0; j<signals; j++)
        if(edfplus)
            skip = 0;
            for(p=0; p<nr_annot_chns; p++)
                if(j==annot_ch[p])
                    skip = 1;
                    break;
    if(skip) continue;

    d_tmp = edfparam[j].smp_written * edfparam[j].time_step;
    if(d_tmp<time_tmp) time_tmp = d_tmp;
    fprintf(outputfile, "%.16f", elapsedtime + time_tmp);
    for(j=0; j<signals; j++)
        if(edfplus)
            skip = 0;
            for(p=0; p<nr_annot_chns; p++)
                if(j==annot_ch[p])
                    skip = 1;
                    break;
    if(skip) continue;

    d_tmp = edfparam[j].smp_written * edfparam[j].time_step;

    // conversion from ascii character to raw values
    if((d_tmp<(time_tmp+0.00000000000001))&&(d_tmp>(time_tmp-
        0.00000000000001))&&(edfparam[j].smp_written<edfparam[j].smp_per_record))
        value_tmp = (((signed short *)cnv_buf+edfparam[j].buf_offset+edfparam[j].smp_written)) + edfparam[j].offset) *
        edfparam[j].sense;
        dynamicArray[(i*recordsize)+edfparam[j].smp_written] = value_tmp; //put value_tmp to dynamicArray
        edfparam[j].smp_written++;
} else fputc(',', outputfile);
}

if(fputc('n', outputfile)==EOF)
{
    printf("Error when writing to outputfile during conversion\n");
    fclose(inputfile);
    fclose(annotationfile);
    fclose(outputfile);
    free(edf_hdr);
    free(edfparam);
    free(cnv_buf);
    free(time_in_txt);
    free(duration_in_txt);
    free(scratchpad);
    return(1);
}

recordfull = 1;
for(j=0; j<signals; j++)
{
    if(edfparam[j].smp_written<edfparam[j].smp_per_record)
    {
        if(edfplus)
        {
            skip = 0;

            for(p=0; p<nr_annot_chns; p++)
            {
                if(j==annot_ch[p])
                {
                    skip = 1;
                    break;
                }
            }
        }
        if(skip) continue;
    }

    recordfull = 0;
    break;
}
}
while(!recordfull);
datarecordswritten++;
}

int numArrays = 0;
numArrays = (datarecords*recordsize)/512;

//Initialize 2-dimensional array512
complex<float> **array512 = new complex<float>*[numArrays];

//Initialize sub-arrays of array512 with 512 block-size
for(int i = 0; i < numArrays; i++)
{
    array512[i] = new complex<float>[512];
}

//Copy contents of dynamicArray to array512
int ctr = 0;
for(int j=0;j<numArrays && ctr<numSamples; j++)
{
    for(int k=0;k<512 && ctr<numSamples; k++)
    {
        array512[j][k] = dynamicArray[ctr];
        ctr++;
    }
}
//Perform FFT for each 512-size block
ofstream outStream;
outStream.open("fft.csv");

complex<float> *newArray = new complex<float>[512];

float incfreq = (recordsize/data_record_duration)/512.000;
float frequency = 0.000;
for(int i=0;i<numArrays;i++){
    newArray = fft(512,array512[i]);
    outStream << "Array " << i+1 << "n";
    for(int j=0;j<256;j++){
        frequency = incfreq*j;
        outStream << frequency << ",";
        outStream << abs(newArray[j]) << "n";
    }
}
outStream.close();

//delete dynamic arrays;
delete [] newArray;
delete [] dynamicArray;
for(int i=0;i<numArrays;i++){
    delete [] array512[i];
}
delete [] array512;

close(inputfile);
close(annotationfile);
close(outputfile);
free(edf_hdr);
free(edfparam);
free(cnv_buf);
free(time_in_txt);
free(duration_in_txt);
free(scratchpad);
remove(ascii_path);

system("PAUSE");
return(0);
A1.2. fourier.h

#ifndef FOURIER_FUNCTIONS
#define FOURIER_FUNCTIONS

#include <iostream>
#include <complex>
#include <cmath>
using namespace std;

const float PI = atan(1)*4;

complex<float> *dft(int N, complex<float> *f)
{
  complex<float> *F = new complex<float>[N];
  complex<float> e;
  for(int k=0; k<N; k++)
  {
    F[k] = (0,0);
    for(int n=0; n<N; n++)
    {
      e = complex<float>(cos(2*PI*k*n/N),sin(-2*PI*k*n/N)); //compute exponential factor
      F[k] = F[k] + f[n]*e;
    }
  }
  return F;
}

complex<float> *fft_rec(int N, complex<float> *f)
{
  int Nprime,n, kprime;
  complex<float> *fe = new complex<float>[(N/2)];
  complex<float> *fo = new complex<float>[(N/2)];
  complex<float> *Fe = new complex<float>[(N/2)];
  complex<float> *Fo = new complex<float>[(N/2)];
  complex<float> *F = new complex<float>[N];
  complex<float> twiddle;
  if(N==1){
    return f; //trivial if N==1
  }else{ //perform 2 sub-transforms
    Nprime = N/2; //size of sub-transforms
    for(int n=0; n<Nprime; n++) //perform N DIF 'butterflies'
    {
      twiddle = complex<float>(cos(2*PI*n/N),sin(-2*PI*n/N)); //compute exponential factor
      fe[n] = f[n]+f[n+Nprime]; //even subset
      fo[n] = (f[n]-f[n+Nprime])*twiddle; //odd subset
    }
    Fe = fft_rec(Nprime,fe); //even k
    Fo = fft_rec(Nprime,fo); //odd k
    for(int kprime=0; kprime<Nprime; kprime++)
    {
      F[2*kprime] = Fe[kprime]; //even k
      F[2*kprime+1] = Fo[kprime]; //odd k
    }
    return F;
  }
}

complex<float> *fft_nrec(int N, complex<float> *f)
{
  int d;
  int M = N/2;
  if(N%2>0){M+=1;}
  float w,z;
  complex <float> DFTe(0,0);
  complex <float> DFTo(0,0);
  complex <float> *F = new complex<float>[N];
  complex <float> e(0,0);

complex <float> i(0,1);
complex <float> g(0,0);

for(int k=0; k<N; k++)
{
    w = (-2.00*PI)/M;
    DFTe = complex<float> (0,0);
    DFTo = complex<float> (0,0);
    for(int m=0; m<M; m++)
    {
        e = exp(w*m*k*i);
        d = 2*m;
        if(m==0) { DFTe = f[d]*e; }
        else   { DFTe = DFTe + f[d]*e; }
    }
    for(int m=0; (2*M==N) && m<M; m++)
    {
        e = exp(w*m*k*i);
        d=2*m+1;
        if(m==0) { DFTo = f[d]*e; }
        else   { DFTo = DFTo + f[d]*e; }
    }
    for(int m=0; (2*M!=N) && m<(M-1); m++)
    {
        e = exp(w*m*k*i);
        d = 2*m+1;
        if(m==0) { DFTo = f[d]*e; }
        else   { DFTo = DFTo + f[d]*e; }
    }
    z = (-2.00*PI)/N;
    g = exp(z*k*i);
    F[k] = DFTe + (DFTo*g);
}

return F;

#endif
A1.3. errorcompute.cpp

```cpp
#include <iostream>
#include <fstream>
#include <complex>
#include "fourier.h"
using namespace std;

int main(){
    ifstream inStream;
    ofstream outStream;

    int N = 512;
    complex<float> *array = new complex<float>[N];
    inStream.open("data.csv");
    for(int i=0;i<N;i++){
        inStream >> array[i];
        //cout << array[i] << "\n";
    }
    complex<float> *dftArray = new complex<float>[N];
    complex<float> *fftRecArray = new complex<float>[N];
    complex<float> *fftNrecArray = new complex<float>[N];
    dftArray = dft(N,array);
    fftRecArray = fft_rec(N,array);
    fftNrecArray = fft_nrec(N,array);
    outStream.open("output.csv");
    outStream << "DFT Definition,Recursive FFT, Percent Difference, ,DFT Definition,Non-Recursive FFT,Percentage Difference\n";
    for(int c=0;c<N;c++)
    {
        outStream << abs(dftArray[c]) << "," << abs(fftRecArray[c]) << "," << floor((abs(fftRecArray[c])-
                        abs(dftArray[c]))*100+0.5)/100;
        outStream << "," << abs(fftNrecArray[c]) << "," << floor((abs(fftNrecArray[c])-
                        abs(dftArray[c]))*100+0.5)/100;
        outStream << "\n";
    }
    inStream.close();
    outStream.close();
    system("PAUSE");
    return 0;
}
```
Appendix 2: Original Project Proposal

Michael Abay, Juan Wilfredo Ibañez and Julia Segismundo
V-BSECE

Main topic: Frequency Domain Analysis (FFT)

Step 1: Project Title
Frequency Domain Analysis of an ECG Signal

Step 2: Expected Outcome
Given an ECG signal in EDF format, we want to produce its frequency domain equivalent. Furthermore, we want the program to determine the highest frequency content of the signal which can be useful in the analysis of whether or not the signal is normal or not, based on certain studies. Our group is also exploring on the possibility of having the output transform in binary format.

Shown below are sample graphs based from our previous analysis using SigView.

Fig. 1: Sample 10-sec ECG signal (WAV).
Step 3: Algorithm/Implementation:

Our entire program would have three major parts:

a. Input of the File
b. FFT Performance
c. Output of the result

The program would allow the user to access the original ECG signal which is in EDF form. After that, it will also determine the actual length of ECG signal to be processed for analysis.

In processing the FFT of the entire signal, the program will also divide it into segments of signals in such a way that the program would not encounter a problem in dealing with a large file of ECG signals. The program will then get the average FFT of all the segments of the original ECG signal which will serve as the basis for analysis. The algorithm that we are planning to implement for our program is radix-4 FFT. The result will be significant in the study of the various ECG signals.

In terms of the output, the results will be saved into CSV file that would be read by a spreadsheet. Furthermore, it can also be graphed using excel application. The output can be cross-referenced with the ECG signals of a database, in this case the Physionet.
Library, where from which a cardiologist or a heart specialist can determine whether a test subject's ECG result is within normal cardiovascular performance or at a certain level of irregular heart beat. In doing so, there are classes to be created in our C++ programming codes for implementation.

**Step 4: Test**

The program may be tested using ECG files available from the Physionet Library. A sample signal can be prepared from an EDF file retrieved from Physionet. We then process the original signal using our program which then outputs a file in a binary format. The output file can be compared to the FFT done to the original ECG sample using programs such as Sigview or Matlab. The program also determines the frequency with the highest content. This can also be verified using Sigview or Matlab.

**Step 5: User Interface**

User operation is done using MSDOS. At the start of the program, a message prompts the user to input the file name of the ECG signal in EDF form. When the user presses enter, the program shall commence the FFT analysis. It will then create another file for the FFT output. After the creating the file, the program will print a message telling the user the frequency of highest content. The program may ask if the user wants to open another file, repeating the process.

**Step 6: References**

Absolute C++ Programming by Walter Savitch
http://www.physionet.org/physiobank/database/#ecg
http://www.edfplus.info/specs/edf.html
Bibliography


